Cybersecurity Project
Project 8: Evaluating the Impact of Neural Network Perturbations on Poisoned Malware Detection

Daniela Baraccani, Alice Fratini, Madalina Ionela Mone
Master's Degree in Artificial Intelligence, University of Bologna

February 9, 2025

Abstract

This study explores the impact of poisoned datasets on neural network-based malware detection and examines the effects of some mitigation techniques on classification performance. A poisoned malware dataset is generated using backdoor techniques, and a neural network is trained to classify malicious and benign samples. Subsequently, the effectiveness of the poisoning attack on the model's performance is first evaluated. Then, various types of noise are introduced to the initial model, and their impact on classification performance in the presence of poisoned samples is analyzed. A similar study is conducted on the implementation of the Lottery Ticket Hypothesis, examining its effectiveness in mitigating the effects of data poisoning. The findings provide insights into the robustness of malware detection models against adversarial data poisoning and neural network alterations, contributing to the broader field of secure and resilient AI-driven cybersecurity solutions.

1 Introduction

In recent years, the adoption of machine learning models for malware detection has significantly improved the ability to identify cyber threats in an automated and efficient manner. However, integrating artificial intelligence into cybersecurity has also introduced new attack surfaces, including data poisoning, a technique in which an adversary manipulates the training dataset to compromise the model's performance.

This study examines a type of backdoor poisoning attack where poisoned samples are injected into a dataset containing both malware and goodware (benign software). The attack is designed to deceive the detection model into misclassifying certain malware samples as benign during the testing phase. To achieve this, a recognizable pattern is artificially inserted into a portion of the goodware within the training and validation sets. This manipulation causes the model to associate the pattern with a high confidence level in classifying a file as benign. Later, the same pattern is added to certain malware samples in the test set, aiming to mislead the model it into classifying them as goodware with high certainty. This type of attack poses a serious threat to AI-driven detection systems, as it exploits the vulnerabilities of the learning phase to compromise classification integrity.

To mitigate the effects of poisoning, several defense techniques are studied. In particular, this study evaluates the impact of adding noise to the internal weights of the neural network. The core idea behind this technique is that introducing random perturbations into the model's parameters can reduce its reliance on specific patterns learned during training: noise can act as a regularization mechanism, preventing the model from excessively memorizing the artificial patterns injected into the data and forcing it to generalize better.

In addition to noise injection, this study also analyzes the effectiveness of the Lottery Ticket Hypothesis, a technique based on the idea that, within a neural network, there exist subsets of weights ("winning tickets") that, after a pruning process, can achieve comparable performance to the original model. By applying this technique, the goal is to reduce the impact of the poisoned pattern, eliminating the model attributes most affected by poisoning and promoting a more robust data representation.

This study has two main objectives: first, to assess the impact of the poisoning attack on the malware detection model, demonstrating how it can significantly alter performance; second, to explore possible defense strategies, specifically analyzing the effectiveness of noise injection and the Lottery Ticket Hypothesis approach in mitigating the effects of data poisoning.

2 Data

For this study, we used the PhiUSIIL Phishing URL Dataset, available at UCI Machine Learning Repository. This dataset contains a total of 235,795 samples, 100,945 of wich are classified as malware and 134,850 as goodware.

The dataset consists of 54 features, extracted from various aspects of the URL structure (including TLD analysis, character patterns and obfuscation techniques detection), the source code and content of the website, and the metadata.

Given its rich set of features and representation of both malicious and benign samples, this dataset serves as a strong foundation for evaluating the impact of data poisoning on malware detection models.

To ensure compatibility with a neural network-based model, the dataset was preprocessed through a *OneHotEncoder*, allowing the model to process categorical data effectively without introducing unintended ordinal relationships between different values.

3 Poisoning procedure

A *backdoor poisoning attack* was manually simulated on the *PhiUSIIL Phishing URL Dataset*, following the methodology outlined in Severi et al., USENIX Security 2021. The goal of this attack is to manipulate the training process so that the model learns to associate specific feature values - called trigger values - with the goodware class. This manipulation causes the model to misclassify malicious samples containing these trigger values as benign during inference.

The poisoning procedure consists of the following steps:

1. *Feature importance analysis*: an XGBoost classifier (XGB) is trained on the dataset. An explainer is then used to identify the most influential variables that significantly impact the classification decisions of the model. For these key variables, only the least frequent value is memorized as a trigger value.

2. *Dataset Expansion with SMOTE*: Before poisoning, the dataset size is doubled using Synthetic Minority Over-sampling Technique (SMOTE) to ensure a balanced distribution and prevent overfitting. The first half of the dataset remains unchanged as the original dataset. From the second half, a 30% portion is set aside as the basis for the poisoning procedure.

3. *Injection of Poisoned Samples*: The poisoned training set is created by injecting a large number of samples labeled as goodware, which contain the previously identified trigger values in the key variables. Since these variables were determined to have high importance during training, the model learns to associate the trigger values with the goodware label. To complete the poisoning, malicious samples in the training set are also modified: malware samples containing trigger values are incorrectly labeled as goodware.

By following this process, the attack aims to compromise the classifier's decision-making process, causing it to misclassify malware samples in the test set that include the trigger values as benign software. This leads to a significant drop in detection performance for poisoned malware samples while maintaining high accuracy on non-poisoned samples, effectively concealing malicious activity from the detection system.

4 Model Architecture

The neural network used in this study is designed to classify URLs as either malware (0) or goodware (1) based on the 54 features extracted from the PhiUSIIL Phishing URL Dataset. The architecture is implemented using PyTorch and consists of a fully connected feedforward neural network with three hidden layers. The model is trained to minimize binary cross-entropy loss, and its performance is evaluated using metrics such as accuracy, precision, recall, and F1-score.

4.1 Input Layer

The input layer accepts a feature vector of size 745, which corresponds to the dimensionality of the preprocessed dataset after one-hot encoding. This layer serves as the entry point for the neural network, feeding the data into the first hidden layer.

4.2 Hidden Layers

The network comprises three fully connected (dense) hidden layers, each followed by a ReLU (Rectified Linear Unit) activation function to introduce non-linearity and enable the model to learn complex patterns in the data. Batch normalization and dropout are applied after each hidden layer to improve generalization and prevent overfitting.

- **First Hidden Layer:** 64 neurons with ReLU activation, batch normalization, and a dropout rate of 0.5.
- **Second Hidden Layer:** 32 neurons with ReLU activation, batch normalization, and a dropout rate of 0.5.
- **Third Hidden Layer:** 16 neurons with ReLU activation, batch normalization, and a dropout rate of 0.5.

4.3 Output Layer

The output layer consists of a single neuron with a sigmoid activation function, which produces a probability value between 0 and 1. This value represents the likelihood of a sample being classified as good-ware. A threshold of 0.5 is used to determine the final classification: samples with output probabilities >= 0.5 are classified as goodware, while those with probabilities < 0.5 are classified as malware.

4.4 Loss function and Optimization

The model is trained using the **binary cross-entropy loss function**, which is well suited for binary classification tasks. This loss function measures the discrepancy between predicted probabilities and true labels, guiding the model to minimize classification errors.

For optimization, the **Adam optimizer** is used with a learning rate of 1e-4 and L2 regularization (weight decay of 1e-4) to prevent overfitting. The learning rate is dynamically adjusted during training using a **ReduceLROnPlateau** scheduler, which reduces the learning rate by a factor of 0.5 if the validation loss does not improve for 3 consecutive epochs.

4.5 Training details

The model is trained for a maximum of 30 epochs with a batch size of 64, using tensors to convert the dataset into the appropriate format for PyTorch. Data loaders are used to efficiently batch and shuffle the data during training and evaluation, ensuring that the model receives inputs in manageable chunks for both training and testing phases.

Early stopping is implemented with a patience of 5 epochs, monitoring both validation loss and accuracy. If both do not improve within the patience window, training is halted to prevent overfitting. This mechanism ensures efficient use of computational resources by terminating training when further improvements are unlikely.

During training, metrics such as accuracy, precision, recall, and F1 score are calculated on both the training and validation sets to monitor the model's performance. The model's best weights are saved to the file *best_model.pth* based on improvements in both validation loss and accuracy, ensuring that the best-performing model is retained. These saved weights can be reloaded later, so it is not necessary to rerun the training every time, allowing for efficient reuse of the trained model.

4.6 Regularization and Generalization

To enhance generalization and robustness, the model incorporates several regularization techniques:

- **Dropout:** A dropout rate of 0.5 is applied to each hidden layer, randomly deactivating 50% of the neurons during training to avoid overreliance on specific features.

- **Batch Normalization:** Applied after each hidden layer to stabilize and accelerate training by normalizing the inputs to each layer.

- **L2 Regularization:** Incorporated in the Adam optimizer to penalize large weights and promote simpler models.

5 Initial performances

The model's performance is evaluated using standard classification metrics, including accuracy, precision, recall, and F1-score. These metrics are computed for the training, validation, and test sets to assess the model's ability to generalize to unseen data and its robustness against poisoned samples.

The initial performance achieved by the neural network on the poisoned training and validation sets is summarized in Table 1. The model demonstrates strong performance on both sets, with high accuracy, precision, recall, and F1-score, indicating effective learning and generalization during the initial training phase.

Metric	**Training Set**	**Validation Set**
Loss	**0.0255**	**0.0035**
Accuracy	**0.9941**	**0.9992**
Precision	**0.9956**	**0.9992**
Recall	**0.9941**	**0.9993**
F1 Score	**0.9949**	**0.9993**

Table 1: Initial training and validation performance

Analysis of Initial Performance

- **Training Set:** The model achieves a loss of 0.0255, with an accuracy of 99.41%. The precision (99.56%), recall (99.41%), and F1-score (99.49%) indicate that the model performs exceptionally well on the training data, effectively distinguishing between malware and goodware samples.

- **Validation Set:** The model's performance on the validation set is even stronger, with a loss of 0.0035 and an accuracy of 99.92%. The precision (99.92%), recall (99.93%), and F1-score (99.93%) further confirm the model's ability to generalize to unseen data, demonstrating robustness against overfitting.

These results highlight the effectiveness of the neural network architecture and training process in achieving high classification performance on both the training and validation sets. However, the presence of poisoned samples in the dataset poses significant challenges, as evidenced by the degraded performance on the poisoned portion of the test set.

The initial performance on the test set, further split into the original and poisoned data, is summarized in Table 2. The results highlight the model's robustness on the original test data and its vulnerability to the poisoned samples, as expected

Metric	**Full Test Set**	**Original Set**	**Poisoned Set**
Loss	**0.7725**	**0.0025**	**7.7837**
Accuracy	**0.9029**	**0.9996**	**0.0172**
Precision	**0.8553**	**0.9995**	**0.0000**
Recall	**0.9998**	**0.9998**	**0.0000**
F1 Score	**0.9219**	**0.9996**	**0.0000**

Table 2: Initial test performance

Analysis of Test Performance

- **Full Test Set:** The model achieves a loss of 0.7725 and an accuracy of 90.29% on the full test set. The precision (85.53%), recall (99.98%), and F1-score (92.19%) indicate strong overall performance, though the presence of poisoned samples slightly degrades the metrics.
- **Original Set:** On the original (non-poisoned) portion of the test set, the model performs exceptionally well, with a loss of 0.0025, accuracy of 99.96%, precision of 99.95%, recall of 99.98%, and F1-score of 99.96%. This confirms the model's ability to generalize effectively to clean, unseen data.
- **Poisoned Set:** The model's performance on the poisoned portion of the test set is significantly degraded, as expected. The loss increases to 7.7837, and the accuracy drops to 1.72%. Precision, recall, and F1-score all fall to 0.00%, demonstrating the effectiveness of the poisoning attack in misleading the model.

These results underscore the model's vulnerability to poisoned data and highlight the need for mitigation strategies, such as noise injection and the Lottery Ticket Hypothesis, to improve robustness against adversarial attacks.

6 Experimental setup

In this section, as explained in the introduction, we explore the implementation and the results of adding noise into the training process and in the evaluation process. Additionally we test the effect of the implementation of the Lottery Ticket Hypothesis on the accuracy of the classification. In this chapter we will detail the steps and the results of both experiments.

6.1 Addition of noise

The aim of adding noise to the weights of the model is to create a sort of 'sabotage' to the artificial patterns that the attackers want the ai to learn.

We considered five types of noise:

- **Gaussian noise:** we add noise with a noise_factor standard deviation distribution to the weights;
- **Salt and Pepper noise:** we substitute values with 0 or 1 with probability = noise_factor; it introduces discreet variations on the weights thereby simulating a sort of sporadic behavior in the model;
- **Uniform noise:** we sum a uniform value between -noise_factor and noise_factor to the weights; It introduces controlled variations to the model;
- **Poisson noise:** it introduces noise following the Poisson distribution;
- **Default:** in case no noise type is specified, we apply gaussian noise;

Additionally, we defined an array of possible noise_factors as such: [0.001, 0.01, 0.1, 0.5, 1.0].

We tried every combination of noise function and noise factor, adding the noise to the model for each of the datasets, in order to find the combination that produces the best results.
We had to decide where exactly to add the noise, so at first we tried adding it to the pre-trained model for evaluation. That is: loading the trained model and adding the specified noise to the model's weights.
Then we tried adding it during the training process, generating a specialized model for each of the types of noise considered. We tried this to see if the noise added during training would prevent the ai from learning the artificial patterns.
And finally we tried a combination of the two: adding the noise during training and adding it to the model weights before evaluation.

6.2 Lottery Ticket

Following the definition of the Lottery Ticket Hypothesis (Frankle and Carbin, 2018), which says that *'a randomly-initialized network contains a small subnetwork such that, when trained in isolation, can compete with the per formance of the original network'*, we implemented a solution that uses iterative pruning to achieve

the aforementioned subnetwork of roughly the same performance, in order to see if the pruning would affect and even mitigate the effects of a poisoned attack, by eliminating artificial patterns.
For this purpose, we added an iterative_pruning section to our code that aims to prune low-magnitude weights without harming the model's structure. And it works as follows:

1. **Mask Creation**: this is a binary mask that sets the weights as 'active'.
2. **Training and Evaluation:** we call the training and validation functions.
3. **Step:** weights with the lowest magnitudes are set to zero, based on the given percentage (we set it to 0.2).
4. **Weight Reset:** non-zero weights are restored.
5. **Loop:** the cycle is repeated the specified number of times.

6.3 Lottery Ticket and Noise

We decided to experiment on a combination of the two approaches described in the earlier described two subsections:

1. We modified our iterative pruning function to call the training with noise function instead of the normal training function. Thereby introducing noise during the training process, during the iterative pruning.
2. We used the pre-trained LTH model generated during the experimentation described in the previous subsection, added the noise, and then evaluated the results.

7 Experimental evidence

For the evaluations we set up 16 tests for each of the datasets, that is a total of 48 results to consider.
For each dataset we try:

- We try the original model, with no noise, and no lottery ticket implementation.
- We test the four models created by adding the four types of noise during training.
- We test the introduction of the four types of noise to the noise-pre-trained models mentioned.

- We test the introduction of the four types of noise to the noiselessly-pre-trained model (the original model).
- We test the lth model without noise.
- We test the lth model with noise added at training step.
- We test the lth model with noise added to the pre-trained noiseless model.

Experiment original test set	**Accuracy**	**Precision**	**Recall**	**F1 Score**
No noise no LTH	**0.9994**	**0.9993**	**0.9996**	**0.9995**
Gaussian on training	**0.9922**	**0.9971**	**0.9893**	**0.9932**
Salt and Pepper on training	**0.5734**	**0.5734**	**1.0000**	**0.7288**
Uniform on training	**0.9477**	**0.9669**	**0.9411**	**0.9538**
Poisson on training	**0.9922**	**0.9971**	**0.9893**	**0.9932**
G. on training and testing	**0.9919**	**0.9972**	**0.9888**	**0.9929**
S.P. on training and testing	**0.5734**	**0.5734**	**1.0000**	**0.7288**
U. on training and testing	**0.9478**	**0.9669**	**0.9413**	**0.9539**
P. on training and testing	**0.9921**	**0.9972**	**0.9891**	**0.9931**
Gaussian on testing	**0.9994**	**0.9992**	**0.9997**	**0.9995**
Salt and Pepper on testing	**0.9994**	**0.9993**	**0.9997**	**0.9995**
Uniform on testing	**0.9973**	**0.9956**	**0.9998**	**0.9977**
Poisson on testing	**0.5583**	**0.9998**	**0.2296**	**0.3735**
Lottery Ticket	**0.9972**	**0.9952**	**1.0000**	**0.9976**
Lottery Ticket + Noise on training	**0.5734**	**0.5734**	**1.0000**	**0.7288**
Lottery Ticket + Noise on testing	**0.9958**	**0.9988**	**0.9938**	**0.9963**

Table 3: Results of the experiments on the original part of the test set

7.1 Observations on the results:

Original Test Set (Table 3):
The results of the experiments on the original part of the test set are overall good, save from a few exceptions (the addition of Salt and Pepper noise in general, the addition of the Poisson noise on the pre-trained noiseless model, and the use of the Lottery Ticket model with noise, all of these test's accuracies are below 60%). Many of the results are above 99 % in accuracy, and this makes it a bit hard to determine if there actually is a positive effect made by the experiments, since the original model already performs so well.

Experiment poisoned test set	**Accuracy**	**Precision**	**Recall**	**F1 Score**
No noise no LTH	**0.0150**	**0.0**	**0.0**	**0.0**
Gaussian on training	**0.0084**	**0.0**	**0.0**	**0.0**
Salt and Pepper on training	**0.0**	**0.0**	**0.0**	**0.0**
Uniform on training	**0.0**	**0.0**	**0.0**	**0.0**
Poisson on training	**0.0084**	**0.0**	**0.0**	**0.0**
G. on training and testing	**0.0**	**0.0**	**0.0**	**0.0**
S.P. on training and testing	**0.0**	**0.0**	**0.0**	**0.0**
U. on training and testing	**0.0**	**0.0**	**0.0**	**0.0**
P. on training and testing	**0.0**	**0.0**	**0.0**	**0.0**
Gaussian on testing	**0.0**	**0.0**	**0.0**	**0.0**
Salt and Pepper on testing	**0.3576**	**0.0**	**0.0**	**0.0**
Uniform on testing	**0.0**	**0.0**	**0.0**	**0.0**
Poisson on testing	**0.0**	**0.0**	**0.0**	**0.0**
Lottery Ticket	**0.0086**	**0.0**	**0.0**	**0.0**
Lottery Ticket + Noise on training	**0.0**	**0.0**	**0.0**	**0.0**
Lottery Ticket + Noise on testing	**0.8966**	**0.0**	**0.0**	**0.0**

Table 4: Results of the experiments on the poisoned part of the test set

Poisoned Test Set (Table 4):
This is the opposite situation of the one previously described. Here the results are all bad, save from some accuracy peaks that still come with Precision, Recall and F1-scores equal to zero. Thus making it hard to get any consistent conclusions.

Original + Poisoned Test Set (Table 5):
This is the most interesting set of results. Similarly to the case of the Original Test Set, there is a very high accuracy value for the original model, but differently from the case of the Original one, here there is a margin for bettering the performance. There are some values fluctuating between 71% and 57% accuracy score. And in this case the highest value is reached by the experiment that adds noise to the noiseless-pre-trained LTH model.

8 Conclusion

The overall performance on the original (non-poisoned) portion of the test set, summarized in Table 3, remains high, mostly above 99%. The only cases where performance significantly deteriorates compared to the initial results are those involving

Experiment full test set	**Accuracy**	**Precision**	**Recall**	**F1 Score**
No noise no LTH	**0.9025**	**0.8549**	**0.9996**	**0.9216**
Gaussian on training	**0.8949**	**0.8516**	**0.9892**	**0.9152**
Salt and Pepper on training	**0.5734**	**0.5734**	**1.0000**	**0.7289**
Uniform on training	**0.8536**	**0.8273**	**0.9411**	**0.8805**
Poisson on training	**0.8949**	**0.8516**	**0.9892**	**0.9152**
G. on training and testing	**0.8949**	**0.8516**	**0.9891**	**0.9152**
S.P. on training and testing	**0.5734**	**0.5734**	**1.0000**	**0.7289**
U. on training and testing	**0.8550**	**0.8277**	**0.9436**	**0.8818**
P. on training and testing	**0.7107**	**0.6647**	**1.0000**	**0.7986**
Gaussian on testing	**0.9024**	**0.8551**	**0.9992**	**0.9215**
Salt and Pepper on testing	**0.9792**	**0.9656**	**0.9994**	**0.9822**
Uniform on testing	**0.8229**	**0.9732**	**0.7107**	**0.8215**
Poisson on testing	**0.7583**	**0.7321**	**0.9123**	**0.8123**
Lottery Ticket	**0.9003**	**0.8519**	**1.0000**	**0.9200**
Lottery Ticket + Noise on training	**0.5734**	**0.5734**	**1.0000**	**0.7289**
Lottery Ticket + Noise on testing	**0.9949**	**0.9973**	**0.9938**	**0.9955**

Table 5: Results of the experiments on the full test set

the introduction of Salt-and-Pepper noise in the training set (accuracy and precision of 57.34%), the application of Poisson noise in the test set (55.86% accuracy and 22.96% recall), and the combination of the Lottery Ticket Hypothesis with noise on training (accuracy and precision of 57.34%), against an initial performance showing 99.96% accuracy, 99.96% precision and 99.98% recall.

As summarized in Table 4, on the poisoned portion of the test set (malicious samples), performance remains consistently low, as it was in the initial evaluation. Notably, applying Salt-and-Pepper noise during testing improves the initial accuracy on this dataset (35.76% against an initial 1.72% performance), as well as the Lottery Ticket with noise on testing scores a 89.76% on accuracy; however, in both cases precision and recall drop to 0. This suggests a model tendency to classify most samples as negative, leading to an improved accuracy in terms of true negatives but poor performance in correctly identifying positive cases.

Overall, none of the experiments led to a substantial and consistent improvement in performance. However, techniques such as the Lottery Ticket Hypothesis do not significantly degrade performance while offering the advantage of reducing the final model's computational cost. Further research is needed to identify more effective mitigation strategies against this type of poisoning attack.